From link grammars to categorial grammars.

Erwan Moreau

From link grammars to categorial grammars

Erwan Moreau

LINA - FRE CNRS 2729 - Université de Nantes
2 rue de la Houssinière - BP 92208 - 44322 Nantes cedex 3

Abstract

In this paper, we propose a way to convert link grammars, for which an important English lexicon exists, into categorial grammars. We also provide a parser in order to test the correctness of the approach by comparing its results to the original link grammar parser. Thus our method makes available a lexicon for categorial grammars that covers an important subset of English. The formalism used is very simple, in order that it can be easily reused with other categorial grammars formalisms.

Key words: Link grammars, classsical categorial grammars, English lexicon, efficient parsing.

1 Introduction

Categorial grammars include various formalisms intended to represent languages. These systems generally offer an elegant and precise way to represent natural languages. They are characterized by their total lexicalization and a close relation between syntax and semantics. Recently, some works have been done to make available linguistic resources in the perspective of real applications of categorial grammars [Moo03], [Hoc03]. In [Hoc03], Hockenmaier shows that a wide-coverage Combinatory Categorial Grammars parser can be built by acquiring an English lexicon from the Penn Treebank.

In this article we explore another way to build a categorial grammars lexicon, by converting link grammars data. We propose an intermediate formalism called *categorial link grammars*, where some important features of categorial grammars are emphasized. We also show how to convert such a grammar into a classical categorial grammar, the most simple categorial grammars formalism.

Email address: Erwan.Moreau@lina.univ-nantes.fr (Erwan Moreau).

And in order to show that this transformation has real applications, we propose an efficient English parser that uses the converted data.

Link grammars are defined by Sleator and Temperley in [ST91]. This is a rather simple formalism which is able to represent in a reliable way natural languages. This can be seen in the modelization that the authors provided for English in this sytem : their grammar deals with most of the linguistic phenomena in English, as it can be verified using their link grammar parser [TSL]. Their work is very interesting for our objectives for the following reasons:

- There exists a close relation between link grammars and categorial grammars: both are totally lexicalized and based on some kind of dependency notion.
- It is possible to express the link grammars model as a small set of rules using unification. This point is essential because several categorial grammars formalisms share this property, and also because it seems to be an advantage for learning applications.
- The authors have carefully documented their parser and the lexicon they use, in a clear and precise way. This is important when thinking about real applications, since it is not rare that good parsers (able to deal with a lot of linguistic aspects) often use some hidden mechanisms that make hard to understand how they work.

The transformation we propose as well as the lexicon built from link grammars data are intended to serve as a basis for other categorial grammars applications. In particular we hope that it will be possible to take benefit from some learning properties in categorial grammars to extend the link grammar lexicon. Existing learning algorithms for categorial grammars make this hypothese rather plausible (see [BP89], [Kan98]).

2 Basic link grammars

Link grammars are a formal system defined by Sleator and Temperley in [ST91]. Informally, a sentence is correct in this system if it is possible to *link* all words according to the links needed by each word, defined in the lexicon. Links represent syntactic relations between words.

A link grammar G is a tuple $\langle \Sigma, \mathcal{C}, \triangleright \rangle$, where Σ is the set of words, $\mathcal{C}$ is the set of *connectors*. A *disjunct* is a pair of lists denoted `d(`L`,`R`)`, where L and R are lists of connectors, and $\triangleright$ is the relation assigning disjuncts to words: $w \triangleright d$ means that the disjunct d can be used with the word w.

Given a sentence $w_1, .., w_n$, a *linkage* is a set of links drawn above the sentence,

each link connecting two words and being labelled with a connector. A linkage is valid if it satisfies the following conditions:

- *Planarity:* all links can be drawn above the words without crossing.
- *Connectivity:* given any couple of words (w_i, w_j) there exists a path of links between them.
- *Ordering:* for each word w_i there exists a disjunct d_i such that $w_i \triangleright d_i$ and d_i is satisfied. A disjunct $d_i =$ `d([`L_1`,..,`L_l`], [`R_1`,..,`R_r`])` is satisfied if for each connector L_j (resp. R_j) there is a link labelled with the name of the connector coming to w_i from the left (resp. the right), and for any couple of connectors L_j, L_k (resp. R_j, R_k) with $j < k$, the words $w_{j'}, w_{k'}$ respectively connected to them verify $j' > k'$ (resp. $j' < k'$) [1] .
- *Exclusion:* No two links may connect the same pair of words.

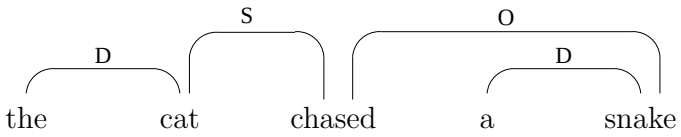
These conditions are called the *meta-rules* of link grammars. It is shown in [ST91] that link grammars are equivalent to context-free grammars.

Example 1 *With the following lexicon, the linkage below is valid:*

a,the ▷ `d([],[D])`

cat, snake ▷ `d([D],[S]), d([O,D],[])`

chased ▷ `d([S],[O])`



3 From link grammars to categorial link grammars

Link grammars are closely related to categorial grammars: both formalisms are totally lexicalized, and share the property to link words or constituents through a binary relation of dependency. However this dependency relation is different in the two systems. In link grammars, dependencies are not directed and apply only to words, contrary to categorial grammars where dependencies are directed and may apply to constituents of the sentence.

[1] In other words when $j < k$ the word $w_{j'}$ is closer to w_i than $w_{k'}$. Remark: in the original definition the right list is written `[`$R_r, \ldots, R_1$`]`. We choose the reverse order to be coherent with the `cons(c,L)/nil` notation presented in section 3.1.

3.1 Categorial link grammars

We propose here an intermediate formalism based on link grammars but closer to categorial grammars. In this formalism, called *categorial link grammars* (CLG), dependencies are directed and apply to constituents. We first show that under some restrictions link grammars can be interpreted with different "meta-rules" without changing their concept. The formalism we propose takes exactly the same definition for grammars as the one presented above. But the rules governing derivations of sentences are expressed as the rewriting of a sequence of terms into another, like in classical categorial grammars.

In order to emphasize that disjuncts are simple terms, we reformulate their definition in the following way: the set of connectors lists CL is defined as the smallest set such that $\texttt{nil} \in CL$ and for all $\texttt{L'} \in CL$ and $\texttt{C} \in \mathcal{C}$, $\texttt{cons(C, L')} \in CL$. The set of disjuncts, now called *types*, is defined as $Tp = \{\texttt{d(L,R)} \mid \texttt{L,R} \in CL\}$. One can see that this definition is coherent with the first one.

3.1.1 Derivations with CLG

The two reduction rules between two types are

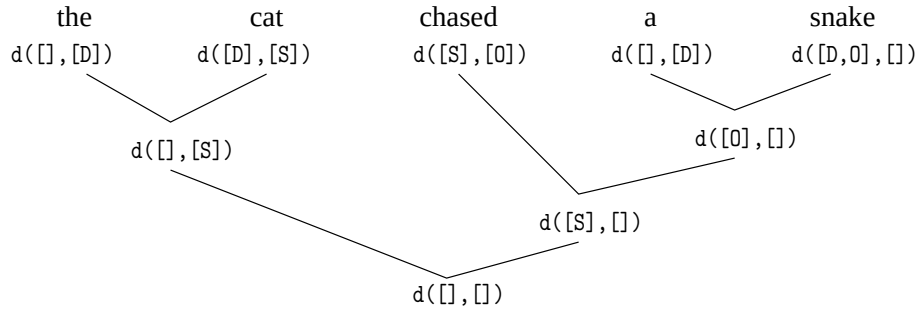
$\texttt{d(L, cons(c, R)), d(cons(c, nil), nil)} \rightarrow \texttt{d(L,R)}$ (with $\texttt{c} \in \mathcal{C}$ and $\texttt{L,R} \in CL$)

$\texttt{d(nil, cons(c, nil)), d(cons(c, L), R)} \rightarrow \texttt{d(L,R)}$ (with $\texttt{c} \in \mathcal{C}$ and $\texttt{L,R} \in CL$)

Let $\Rightarrow$ be the relation defined by $\alpha t_1 t_2 \beta \Rightarrow \alpha t_0 \beta$ if and only if $t_1 t_2 \rightarrow t_0$, with $\alpha, \beta \in Tp^*$ and $t_1, t_2, t_0 \in Tp$. $\Rightarrow^*$ is defined as the reflexive and transitive closure of $\Rightarrow$. Let $G = \langle \Sigma, \mathcal{C}, \triangleright \rangle$ be a grammar. A sentence $x = w_1 w_2 ... w_n$ is correct for G (denoted $x \in L(G)$) if there is a sequence of types $\langle t_1, t_2, ..., t_n \rangle$ such that $w_i \triangleright t_i$ for all i and $t_1 t_2 ... t_n \Rightarrow^* \texttt{d(nil, nil)}$.

Example 2 *With the lexicon defined in example 1, the sentence "The cat chased a snake" can be derived in the following way*[2] *:*

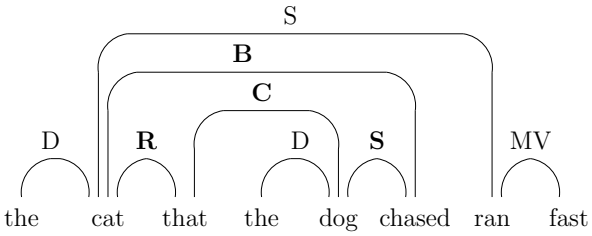
[2] *Remark:* in the following we will keep the notation $[c_1, c_2, .., c_n]$ for connectors lists, easier to read than $\texttt{cons}(c_1, \texttt{cons}(c_2, .. \texttt{cons}(c_n, \texttt{nil}) ..))$. Nevertheless it is important to notice that there is no associativity in these terms.



One can notice that a given linkage in basic link grammars can have several equivalent derivation trees in CLG. These ambiguities appear because links are not directed, and no order is required between connectors.

3.1.2 Equivalence between cycle-free link grammars and CLG

The original definition of link grammars does not forbid cycles. For example, the following linkage where links (R,C,S,B) form a cycle is valid:



A linkage is said *cycle-free* if it does not contain any cycle, and a link grammar G is said cycle-free if for each sentence $x \in L(G)$ there exists a cycle-free linkage.

Proposition 3 *Let $\langle t_1, t_2, .., t_n \rangle$ be a sequence of types, with $n > 1$. The two following propositions are equivalent:*

- *There exists a valid cycle-free linkage for the sequence $\langle t_1, t_2, ..., t_n \rangle$.*
- *There exists two types t_k, t_{k+1} in $\langle t_1, t_2, ..., t_n \rangle$ and a type t' such that $t_k, t_{k+1} \rightarrow t'$ and there exists a valid cycle-free linkage for the sequence $\langle t_1, .., t_{k-1}, t', t_{k+2}, ., t_n \rangle$.*

PROOF.

$\Rightarrow$. Suppose there is a valid cycle-free linkage for $\langle t_1, t_2, .., t_n \rangle$. Since the linkage satisfies the connectivity condition and does not contain any cycle, There must be at least one type t_i that is connected by only one link in the structure. Let t_j be the type to which t_i is connected by this link. If $i < j$, let $l = i$ and $r = j$, otherwise let $l = j$ and $r = i$. We show by induction on the distance

$|r-l|$ that there exists two adjacent types t_k, t_{k+1} in the range $[t_l..t_r]$ which are connected together and such that one of them has only this link:

- if $|r-l| = 1$ then t_i and t_j are adjacent. Since t_i is connected only to t_j, the property holds.
- suppose $|r-l| > 1$. Any link from a type which belongs to the range $[t_l, t_r]$ must be connected to another type inside this range, otherwise this link would cross the link between t_i and t_j and then contradict the planarity condition. Since the linkage is cycle-free, there exists a type $t_{i'}$ which is connected by only one link to a type $t_{j'}$, with $l < i' < j' \leq r$ or $l \leq j' < i' < r$. $|i'-j'| < |r-l|$, so by induction hypothesis there exists two adjacent types t_k, t_{k+1} in the range $[t_{i'}..t_{j'}]$ (or $[t_{j'}..t_{i'}]$ if $j' < i'$) connected together and such that one of them has only this link.

If t_k has only one link going from t_k to t_{k+1} (resp. from t_{k+1} to t_k), then there is a connector `C` such that $t_k =$ `d(nil, cons(C,nil))` and $t_{k+1} =$ `d(cons(C,L),R)` (resp. $t_k =$ `d(L,cons(C,R))` and $t_{k+1} =$ `d(cons(C,nil),nil)`). As a consequence there exists $t' =$ `d(L,R)` such that $t_k, t_{k+1} \rightarrow t'$. t_k (resp. t_{k+1}) is not connected to any other type than t_{k+1} (resp. t_k) and the connectors contained in `L, R` in t_{k+1} (resp. t_k) are the same as those contained in t', therefore it is possible to replace the couple of types (t_k, t_{k+1}) with t' in the linkage: the new linkage obtained for the sequence $\langle t_1, .., t_{k-1}, t', t_{k+2}, ., t_n \rangle$ is valid and cycle-free.

$\Leftarrow$. Suppose there exists a valid cycle-free linkage for the sequence of types $\langle t_1, .., t_{k-1}, t', t_{k+2}, ., t_n \rangle$, and there are two types t_k, t_{k+1} such that $t_k, t_{k+1} \rightarrow t'$. From the form of the rules defining $\rightarrow$ it is clear that one of the two types has only one connector `C`, which can be linked to the other type. The other connectors of the couple of types are the same as those contained in t', so it is possible to replace t' in the linkage with t_k and t_{k+1}, connected together with a link `C`. The linkage obtained for $\langle t_1, .., t_k, t_{k+1}, .., t_n \rangle$ is valid and cycle-free. □

Proposition 4 *Let $\langle t_1, t_2, .., t_n \rangle$ be any sequence of types, with $n \geq 1$. There exists a valid cycle-free linkage for this sequence if and only if $t_1, t_2, .., t_n \Rightarrow^*$* `d(nil,nil)`.

PROOF. By induction on the length n of the sequence:

- if $n = 1$, t_1 must be `d(nil, nil)` because it is the unique valid linkage with only one type. $\Rightarrow^*$ is reflexive.
- if $n > 1$. From proposition 3, there exists a valid cycle-free linkage for $\langle t_1, t_2, .., t_n \rangle$ if and only if there are two adjacent types (t_i, t_{i+1}) $(0 < i < n)$ and a type t' such that $t_i t_{i+1} \rightarrow t'$ (1), and there exists a valid cycle free linkage for $\langle t_1..t_{i-1}t't_{i+2}..t_n \rangle$ (2). The length of this sequence of types is $n-1$, so by induction hypothesis (2) holds if and only if $t_1..t_{i-1}t't_{i+2}..t_n \Rightarrow^*$

`d(nil,nil)`. Since $t_i t_{i+1} \rightarrow t'$ (1), the definition of $\Rightarrow$ gives $t_1 t_2 ... t_n \Rightarrow t_1 .. t_{i-1} t' t_{i+2} .. t_n \Rightarrow^*$ `d(nil,nil)`, which is equivalent to $t_1 t_2 ... t_n \Rightarrow^*$ `d(nil,nil)`.

Corollary 5 *cycle-free link grammars and categorial link grammars are weakly equivalent.*

Clearly any cycle-free linkage is a tree, but there is no isomorphism between this tree and the equivalent derivation tree in CLG. This is due to the fact that nodes in the first tree are words of the sentence, whereas the CLG derivation tree contains types corresponding to constituents of the sentence. However the relation between the two structures is stronger than a simple weak equivalence, because each node in the derivation tree must correspond to a link in the cycle-free linkage.

This similarity between structures is important, because it implies that the meaning of each connector is preserved through the conversion into a CLG grammar. Thus a CLG derivation tree describes real syntactic relations, as they were defined in the original link grammar.

4 Parsing English with a Categorial Link Grammar

One of the main reasons why link grammars are interesting is that the authors Sleator and Temperley have built such a grammar for English. The lexicon they provide contains approximately 59000 words, distributed into 1350 entries, each entry corresponding to a set of types. The grammar deals with an important number of linguistic phenomena in English, *"indicating that the approach may have practical uses as well as linguistic significance"* [ST91]. Furthermore, the link parser provided by the authors includes a file containing 928 sentences, labelled as correct or incorrect.

In this section we explain how it is possible to adapt the link grammars lexicon for English to the formalism of CLG. This adaptation leads to the possibility of parsing a rather large part of English with a CLG, as it will be shown using the example file for link grammars.

4.1 Conversion in practice

First we need to detail some features added to the basis of link grammars by their authors *"to streamline the difficult process of writing the dictionnary"* [ST91]. The choices made to convert the link grammar to a CLG are mainly guided by the necessity to keep the system as simple as possible: thus, the

system will be maximally reusable (in particular for learning applications, see section 6). It is also emphasized that the conversion that we propose is totally automatic: the advantage is that it can be computed with any link grammar, but it is clear that some parts could be improved using the human expertise.

4.1.1 Multi-connectors and the dictionnary format

Iterations are represented using *multi-connectors*: Any connector `C` in a connectors list may be preceded by the operator `@`, meaning that this connector `@C` can be connected to several links `C`. For example, nouns can be given the type `d([@A,D],[S])`, allowing them to be preceded by any number of adjectives. This feature is included in the same way in CLG, by replacing the "basic rules" with these new ones (where `[@]` means that `@` is optionnal):

`d(L, cons([@]c, R)) , d(cons([@]c, nil), nil)` → `d(L,R)`

`d(nil, cons([@]c, nil)) , d(cons([@]c, L), R)` → `d(L,R)`

`d(L, cons(@c, R)) , d(cons([@]c, nil), nil)` → `d(L,cons(@c, R))`

`d(nil, cons([@]c, nil)) , d(cons(@c, L), R)` → `d(cons(@c, L),R)`

The disjuncts are given in the original dictionnary as formulas using or/and operators, where left connectors are denoted `C-` and right connectors `C+`. An operator "`?`" is allowed for optionnal subformulas. The conversion simply puts all formulas in disjunctive form.

4.1.2 Subscripts

Subscripts are used to specialize connectors, in a similar way than feature structures in unification grammars. They are used to make the grammar easier to read and understand. For example, the subscripts `s` and `p` are assigned to nouns (and pronouns, determiners, etc.) to distinguish between singular and plural ones. Practically the subscript is a sequence (eventually empty) of lower case letters that can contain the wildcard `*`. Two connectors match if their subscripts can be unified. For example, the right connector `Spa+` can match a left connector `S-`, `Sp-` or `Spa-`, but not a `Ss-`, `Ssa-` or `Spb-`.

In order to maintain the simplicity of the reduction rules, subscripts are converted into types that do not contain subscripts. This is achieved through a program that classifies all existing connectors (with their subscripts) in such a way that all connectors in a same class match the same set of connectors. When it is possible two classes are merged together, in order to minimize their number. Finally a new type is created for each such class. In the general case, an incoherence can occur with multi-connectors because it is possible that

a single muti-connector match several connectors with different (and potentially incompatible) subscripts However it appears that this is not the case with the set of connectors used in the English grammar: it is sufficiant that multi-connectors be processed first to avoid any ambiguity in the conversion.

4.1.3 Cycles

Cycles are the main problem for this transformation. We propose two different solutions to get round this difficulty. The first one consists in adding appropriate reduction rules:

```
d(nil,cons(x,cons(y,nil))), d(cons(x,nil),cons(z,nil)) → d(nil,cons(z,cons(y,nil)))
d(cons(z,nil),cons(x,nil)), d(cons(x,cons(y,nil)),nil) → d(cons(z,cons(y,nil)),nil)
          d(nil,cons(x,cons(y,nil))), d(cons(x,cons(y,L)), R) → d(L,R)
          d(L, cons(x,cons(y,R))), d(cons(x,cons(y,nil)),nil) → d(L,R)
```

The first two rules introduce transitivity between two types. These rules are similar to composition rules in Combinatory Categorial Grammars [Ste87]: $A/B, B/C \rightarrow A/C$ (Forward Composition). They permit to reduce the bottom part of the cycle. Then one of the last two rules is used to link two connectors in the same time, thus reducing together the up and bottom parts of the cycle. It is important to note that these last rules violate the exclusion constraint, since two words can be linked together by two different connectors. Nonetheless, no sentence was found in the example file where this method causes an error.

The second solution is to eliminate in each possible cycle one link, so that it is possible to use the standard reducing rules. This task is possible because the original link grammar parser uses some particular constraints on cycles to ensure that some incorrect sentences can not be analyzed. That is why a list of "cycle connectors" is available. It seems in the examples file that deleting these connectors from the types (during the conversion) is sufficiant to eliminate any cycle in the linkages. But there are two drawbacks: firstly, nothing ensures that there is no other cases where cycles are needed, even if there is no example indicating that. Secondly, this simple deletion makes the converted grammar overgenerating, because the cycle links were used to exclude some ungrammatical constructions.

4.2 The CLG parser

The correctness of the conversion has been tested by comparing results of parsing the example file using the converted grammar to the ones obtained using

the original link parser. The CLG parser is a standard CYK-like parser running in $o(n^3)$, applying the binary reduction rules defined above. The rules described in [TSL] concerning punctuation, hyphenated words, capitalized words and idioms are followed. Since the number of disjuncts [3] is considerably increased by the conversion, some methods have been implemented to optimize performances. These methods are mainly the same as those used in the original parser, comprising pruning and "fast-matching data structures" (described in [ST91], [GLS95]).

The file `4.0.batch` provided with the original link parser contains 928 sentences, in which 572 are correct and 356 are not. We extensively used this set of examples as a benchmark to test the CLG parser soundness and completness.

The CLG parser does not handle some "high-level" features of the original parser. Our objective being to provide a simple system based on a little set of rules, we did not translate these abilities into the CLG parser, and consequently modified the set of examples. Therefore sentences needing these features (conjonctions, post-processing rules, cost system and unknown words) have been removed from the example file : after this process, a set of 771 sentences is obtained, in which 221 are incorrect.

4.2.1 Results

The current parser is rather slow [4], probably because all possible optimization have not been implemented yet. Two cases have been tested with the benchmark, corresponding to the two possible solutions for the problem of cycles (see 4.1.3):

- If the cycle-free grammar is used (i.e. the grammar in which cycle links have been removed during the conversion), the parsing of the 771 sentences takes approximately 40 minutes. But the overgeneration (due to the deletion of cycle connectors) causes 38 incorrect sentences to be considered correct (the error rate is then 4.9%).
- If the complete grammar is used with the cycle-reduction rules, the 771 sentences are parsed in approximately 46 minutes. All and only correct sentences are parsed successfully.

Example 6 *Here are some sentences taken from the examples file:*

What did John say he thought you should do
**What did John say did he think you should do*

[3] The biggest entry contains 30360 disjuncts. The average number of disjuncts per entry is 2176.
[4] Tests were done on a PIII 1GHz. The parser is coded in SWI Prolog.

To pretend that our program is usable in its current form would be silly
That our program will be immediately accepted is hardly likely
**Is that our program will be accepted likely*
The man there was an attempt to kill died

5 CLG and classical categorial grammars

AB grammars (also called classical categorial grammars) are the most simple formalism in categorial grammars. An AB grammar G is a tuple $\langle \Sigma, Pr, \triangleright \rangle$ where Σ is the set of words and Pr the set of primitive types. The set of types Tp is defined as the smallest set such that $Pr \subseteq Tp$ and $A/B, A\backslash B \in Tp$ for all $A, B \in Tp$. The relation $\triangleright \subseteq \Sigma \times Tp$ assigns one or several types to each word. The relation $\rightarrow$ is defined with the two following universal rules:

$$A/B,\ B \rightarrow A \text{ (for any } A, B \in Tp)$$
$$B,\ B\backslash A \rightarrow A \text{ (for any } A, B \in Tp)$$

Let $\Rightarrow$ be the relation defined by $\alpha t_1 t_2 \beta \Rightarrow \alpha t_0 \beta$ if and only if $t_1 t_2 \rightarrow t_0$, and let $\Rightarrow^*$ be the reflexive and transitive closure of $\Rightarrow$. A sentence $x = w_1 w_2 ... w_n$ is correct for G (denoted $x \in L(G)$) if there is a sequence of types $\langle t_1 t_2 ... t_n \rangle$ such that $w_i \triangleright t_i$ for all i and $t_1 t_2 ... t_n \Rightarrow^* s$.

5.1 From CLG to AB grammars

AB grammars are equivalent to context-free grammars [BHGS60], thus their expressive power is the same as link grammars. We propose here a more straightforward transformation from cycle-free link grammars to AB grammars, through the intermediate formalism of CLG. We have shown that any cycle-free link grammars can be interpreted as a CLG. Let define the following transformation from CLG to AB grammars:

Let $D = \texttt{d([}L_1\texttt{,..,}L_l\texttt{],[}R_1\texttt{,..,}R_r\texttt{])}$,

$$CLG2AB(D) = \left\{ \begin{array}{l} L_1^l \backslash ... \backslash L_{l-1}^l \backslash L_l^l \backslash s / R_r^r / R_{r-1}^r / .../ R_1^r, \\ L_1^l \backslash ... \backslash L_{l-1}^l \backslash L_l^r / R_r^r / R_{r-1}^r / .../ R_1^r, \\ L_1^l \backslash ... \backslash L_{l-1}^l \backslash L_l^l \backslash R_r^l / R_{r-1}^r / .../ R_1^r \end{array} \right\}$$

All types are flat in the converted grammar, i.e. there is no complex argument: given any subtype of the form A/B, B is always a primitive type. we suppose

that the operators / and \ are associative, i.e. $(a\backslash b)/c = a\backslash(b/c)$. Each connector is labelled with a superscript l or r: this is needed to avoid that two left (or two right) connectors be connected together, when one of them is converted as the principal type.

Given a CLG grammar $G = \langle \Sigma, \mathcal{C}, \triangleright \rangle$, the AB grammar $CLG2AB(G)$ is $\langle \Sigma, Pr, \blacktriangleright \rangle$, where $Pr = \mathcal{C} \cup \{s\}$ and $w \blacktriangleright t$ for all t such that $w \triangleright D$ and $t \in CLG2AB(D)$.

Proposition 7 *Let $G = \langle \Sigma, \mathcal{C}, \triangleright \rangle$ be a CLG grammar and $G' = \langle \Sigma, Pr, \blacktriangleright \rangle$ be an AB grammar such that $G' = CLG2AB(G)$.*

For any type $t' \in Tp_{G'}$ and any sequence of words $x \in \Sigma^+$, there exists $t \in Tp_G$ such that $x \Rightarrow_G^* t$ and $t' \in CLG2AB(t)$ if and only if $x \Rightarrow_{G'}^* t'$.

PROOF.

(only the $\Rightarrow$ part is proved. The $\Leftarrow$ part is similar.)

By induction on the height h of the proof tree:

- $h = 0$. $x = \langle w \rangle \Rightarrow_G^* t$ implies that $w \triangleright t$. By definition $w \blacktriangleright t'$ for all $t' \in CLG2AB(t)$, so $x = \langle w \rangle \Rightarrow_{G'}^* t'$.
- $h > 0$. Suppose the rule used at the root is $t_1, t_2 \rightarrow t$ with $t_1 =$ `d(L,cons(c,R))`, $t_2 =$ `d(cons(c,nil), nil)` and $t =$ `d(L,R)`. Let x_1 and x_2 be the leaves of the two subtrees (i.e. $x = x_1 \bullet x_2$ and $x_1 \Rightarrow_G^* t_1$ and $x_2 \Rightarrow_G^* t_2$). By induction hypothesis, any type $t'_1 \in CLG2AB(t_1)$ (resp. $t'_2 \in CLG2AB(t_2)$) is such that $x_1 \Rightarrow_{G'}^* t'_1$ (resp. $x_2 \Rightarrow_{G'}^* t'_2$). Observe that $CLG2AB(t_2) = \{c, s\}$ (because t_2 has no other connectors), and any $t'_1 \in CLG2AB(t_1)$ has c for rightmost argument. From the form of the rules, it is clear that for each $t' \in CLG2AB(t)$ there exists a type $t'_1 \in CLG2AB(t_1)$ such that $t'_1 c \rightarrow t'$. The case where the first subtree is argument (corresponding to the second rule) is symetrical.

□

Corollary 8 *Let G be a CLG grammar. If $G' = CLG2AB(G)$ then $L(G) = L(G')$.*

PROOF.

Suppose $G' = CLG2AB(G)$. Let $x \in \Sigma^+$ be a sequence of words such that $x \in L(G)$: $x \Rightarrow_G^*$ `d([],[])`. $CLG2AB($`d([],[])`$) = \{s\}$, so by proposition 7 $x \Rightarrow_{G'}^* s$, in other words $x \in L(G')$. In the same way, if $x \Rightarrow_{G'}^* s$ then (also

from proposition 7) there exists t such that $x \Rightarrow_G^* t$ and $s \in CLG2AB(t)$. The only way to have $s \in CLG2AB(t)$ is that $t = \texttt{d([],[])}$, so $x \in L(G')$ implies that $x \in L(G')$. We have shown that $L(G) \subseteq L(G')$ and $L(G') \subseteq L(G)$, therefore $L(G) = L(G')$. $\square$

5.2 An AB grammar parser for English

This transformation has been applied to link grammars data for English. A parser has also been implemented, based on the CLG parser.

Usually iterations in AB grammars are obtained using types of the form t/t or $t\backslash t$. But transformating multi-connectors into such types implies the loss of structural similarities between derivation trees and original linkages. That is why link grammars multi-connectors are handled with special "iteration rules" (like in CLG) of the form $A/(@B), B \rightarrow A/(@B)$ and $A/(@B), B \rightarrow A$ (the backward direction is symetrical).

Cycles can not be handled easily using AB grammars. Adding *ad hoc* rules (as shown in 4.1.3) is possible but it is counter-intuitive in the viewpoint of the distinction functor/argument. For this reason, the current version of the parser only works like the cycle-free version of the CLG parser, i.e. when all cycle-links have been removed during the conversion. As a consequence it is unable to fail (as it should) on 38 incorrect sentences (the error rate is 4.9%).

6 Future work

Clearly the grammar built from this conversion could be improved: conjonctions are not handled although it is a very important point in order to consider real applications with unrestricted texts. The cycles problem is not solved in a totally satisfying way, in particular in the case of AB grammars. But there are also some aspects that need further attention even if it seems to work well: the "double direction" of links in the original lexicon causes unnecessary ambiguities in the derivation trees. It is not clear wether it is possible to set a unique direction for a connector, but perhaps it is possible for some of them. More generally, the lexicon could be improved (manually or not) to take benefit of some more sophisticated categorial grammmars formalisms.

There exists good results about learning categorial grammars in Gold's model [Gol67]: Buszkowski has proposed in [BP89] the first learning algorithm for AB grammars. This algorithm learns rigid AB grammars from structures, and has been extended to several other cases by Kanazawa. In particular,

Kanazawa shows that k-valued AB grammars (in which each word has at most k types) are learnable from flat strings [Kan98]. One of the important points in Kanazawa's work is that the unification method used in Buszkowski's algorithm is general enough to be used in other frameworks. That is why this kind of symbolic learning has been applied to different formalisms (see for example [BR01]).

Recently Bechet has shown that k-valued link grammars are learnable from flat strings [Bec03], and proposed an algorithm. Thus it is highly probable that learning algorithms described by Kanazawa can be adapted to the CLG formalism. In particular on can use the English lexicon built from link grammars:

- to build a non-trivial set of positive examples with structures, in order to test algorithms that learn from structures.
- to test algorithms that learn from strings, by comparing the types they infer from a set of sentences (typically the example files of link grammars) to the lexicon.
- *partial learning:* in the framework of partial learning it is supposed that some words are already known. The goal of the learning algorithm is then to find types for unknown words according to sentences provided as examples, in such a way that they do not contradict with the known part of the lexicon. The link grammars lexicon seems suitable for this case, since it can be used as the known part of the grammar.

Applying learning algorithms to categorial link grammars is particularly interesting, because it would permit to extend the original English lexicon (as well as any other). Actually, in the hypothese where it is possible to design efficient algorithms that learn from simple text, one could obtain a more flexible system by enriching the lexicon with any specialized vocabulary.

7 Conclusion

We have shown how it is possible to convert a link grammar into two different categorial grammars systems. The system we obtain needs some improvements, and is certainly less powerfull than Hockenmaier's one [Hoc03]. Nevertheless, its simplicity (simple AB grammars) together with its ability to deal with an important subset of the English language can give rise to numerous extensions. In particular, the lexicon and rules can easily be extended to suit to other categorial grammars formalisms. Furthermore, closeness between the two systems should permit to combine the learning properties from categorial grammars and the data from link grammars. Thus it would be possible to extend the English lexicon using symbolic learning algorithms, then obtaining

robust and reliable categorial grammars data.

References

[Bec03] Denis Bechet. k-valued link grammars are learnable from strings. In *Proceedings Formal Grammars 2003*, pages 9–18, 2003.

[BHGS60] Y. Bar-Hillel, C. Gaifman, and E. Shamir. On categorial and phrase structure grammars, 1960.

[BP89] Wojciech Buszkowski and Gerald Penn. Categorial grammars determined from linguistic data by unification. Technical Report TR-89-05, Department of Computer Science, University of Chicago, June 1989.

[BR01] Roberto Bonato and Christian Retoré. Learning rigid lambek grammars and minimalist grammars from structured sentences. In Luboš Popelínský and Miloslav Nepil, editors, *Proceedings of the 3d Workshop on Learning Language in Logic*, pages 23–34, Strasbourg, France, September 2001.

[GLS95] Dennis Grinberg, John Lafferty, and Daniel Sleator. A robust parsing algorithm for LINK grammars. Technical Report CMU-CS-TR-95-125, Carnegie Mellon University, Pittsburgh, PA, 1995.

[Gol67] E.M. Gold. Language identification in the limit. *Information and control*, 10:447–474, March 1967.

[Hoc03] Julia Hockenmaier. *Data and models for statistical parsing with Combinatory Categorial Grammar*. PhD thesis, School of Informatics, The University of Edinburgh, 2003.

[Kan98] Makoto Kanazawa. *Learnable classes of categorial grammars*. Cambridge University Press, 1998.

[Moo03] Richard Moot. Parsing corpus-induced type-logical grammars. In R. Bernardi and M. Moortgat, editors, *Proceedings of the CoLogNet/ElsNet Workshop on Linguistic Corpora and Logic Based Grammar Formalisms*, 2003.

[ST91] Daniel D. K. Sleator and Davy Temperley. Parsing english with a link grammar. Technical Report CMU-CS-TR-91-126, Carnegie Mellon University, Pittsburgh, PA, 1991.

[Ste87] Mark Steedman. Combinatory grammars and parasitic gaps. *Natural Language and Linguistic Theory*, 5:403–439, 1987.

[TSL] Davy Temperley, Daniel Sleator, and John Lafferty. Link grammar. http://hyper.link.cs.cmu.edu/link/.